NULLABILITY AND PATTERN MATCHING

By: Benjamin Michaelis

@BenjaminMichaelis

What is nullability and why does it matter

- Often developers assume that reference types accept non-null and null both and there wasn't any explicit handling required and unfortunately this consideration is one of the primary root causes of famous "Null Reference Exception".
- This NullReferenceException is thrown by the program whenever a developer attempts to access any type which has value as null.
- This means we try to access a value or reference that holds no value or a null value.

What is a nullable value type?

- Have been available since C# 2.0
- A nullable value type holds all of its possible values and also null.
  - Ex: you can assign any of the following three values to a bool? variable: true, false, or null
- When do you use this signature?
  - When a variable value could be undefined or missing.
  - For example, a database field in an application may contain true or false, or it may contain no value at all, that is, null. You can use the bool? type in that scenario

MSDN Documentation: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types

Nullable value type?

Examples:

```
int? n = null;
// int m1 = n; // Doesn't compile
// int m2 = null; // Doesn't compile
int n2 = (int)n; // Compiles, but throws an exception if n is null
// n2 = null; // Doesn’t compile
```

MSDN Documentation on nullable value types: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types

What is a nullable reference type?

- Available beginning in C# 8.0 when you opt into nullable aware context using build settings.
- It means quite as it sounds, allowing a reference type such as `string name;` to contain a null value such as `string name = null;`

- MSDN Documentation on nullable reference types: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-reference-types

Nullable reference type examples

```
string description = "widget";
var item = new ProductDescription(description);
string shortDescription = default; // Warning; non-nullable set to null;
var product = new ProductDescription(shortDescription); // Warning; static analysis knows
shortDescription maybe null.
```

- MSDN Documentation: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-reference-types

Nullable reference type examples

```
// Warning: CS8714 Nullability of type argument 'string?' doesn't match `notnu
ll` constraint
var d1 = new Dictionary<string?, string>(10);

// And as expected, using 'null' as a key for a non-
nullable key type is a warning...
var d2 = new Dictionary<string, string>(10);

// Warning: CS8625 - Cannot convert to non-nullable reference type.
var nothing = d2[null];
```

Syntax to know

```
#nullable enable
string notNull = "Hello";
string? nullable = default;
string? nullable2 = null;
notNull = nullable!; // null forgiving operator (Should be avoided)
// It only turns off compiler checks at runtime even when the value may still be null, so only should be used in a rare case when the compiler is not able to detect that a nullable value is actually non-nullable
#nullable restore
```

MSDN Documentation on #nullable: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/preprocessor-directives

Syntax to know

- Null Coalescing Assignment
  - The null-coalescing operator ?? returns the value of its left-hand operand if it isn't null; otherwise, it evaluates the right-hand operand and returns its result.
  - Support a??=b in place of if(a==null) a=b

```
public double Evaluate()
{
        return this.rootNode?.Evaluate() ?? 0.0f;
}
```

  - MSDN Documentation on null coalesce opeator: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator
- { } is not null

How to enable nullable aware context

Nullable context is set in csproj file as follows

```
<LangVersion>8.0</LangVersion>
<Nullable>enable</Nullable>
```

Or in the source code with

```
#nullable enable
#nullable restore
```

Note that C# 8.0 is not meant for older targets, such as .NET Core 2.x or .NET Framework 4.x. So some additional language features may not work unless you are targeting .NET Core 3.0 or .NET Standard 2.1

Documentation: https://devblogs.microsoft.com/dotnet/try-out-nullable-reference-types/?WT.mc_id=DT-MVP-5003978

Intro to pattern matching examples

- Patterns test that a value has a certain type, and can extract information from the value when it has the matching type. Pattern matching provides more concise syntax for algorithms.

```
if (shape is Square)
{
var s = (Square)shape;
return s.Side * s.Side;
}
else if (shape is Circle)
{
var c = (Circle)shape;
return c.Radius * c.Radius * Math.PI;
}
```

- MSDN Documentation on pattern matching: https://docs.microsoft.com/en-us/dotnet/csharp/pattern-matching

Intro to pattern matching examples

- Here we can use a reference word, in this case "item" to refer to an object that we want to select or perform operations on with Enumerables

```
private IEnumerable<Shape> Shapes
{
   get
   {
         return this._sequences.SelectMany(item => item).ToList();
   }
}
```

- MSDN Documentation on enumerators: https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerator

Intro to pattern matching examples

- We can further use those keys words and enumerators to simplify methods significantly

```
private static IEnumerable<Type> GetShapes() =>
AppDomain.CurrentDomain.GetAssemblies().SelectMany(item =>
item.GetTypes().Where(type => type.IsSubclassOf(typeof(Shape))));


public Dictionary<char, Type> ShapeTypes { get; } =
GetShapes().ToDictionary(item => char.ToLower(item.Name[0]));
```

Intro to pattern matching examples

- condition_expression ? first_expression : second_expression;
- With conditional operator ?: if condition_expression is true then it will return first expression, else if false second expression will be returned
  - is this condition true ? yes : no

MSDN Documentation: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator

Enhanced Pattern Matching

- Should be our last resort. Figure out a way to do polymorphism first. If we add a new type, our example switch statement will fail. Using where clauses, filter expression, etc. we can do this already, but this example has better scenarios.

Enhanced Pattern Matching

```
return switch (person)
{
    Professor(_, var lastName, var subject) item =>
        $"Dr. {lastName} teaching {subject}",
    Student { FirstName: var firstName,
        Advisor { LastName: var advisorLastName } } =>
            $"",
    (string firstName, _)
        { EnrollmentStatus: EnrollmentStatus.Enrolled } =>
            $"",
    { } => "Enrollment has passed. See you next year.",
    null => "Oh No!!"
}
```